Shell Script

Theeba

What is UNIX?

- UNIX is a computer operating system
- An operating system is the program; it manages system resources such as processor, secondary memory and I/O devices on behalf of its users. It allocates the computer's resources and schedules tasks.
- UNIX is a multiuser, multiprocessing portable operating system.
- UNIX operating system, developed in the 1970s at the AT &T Bell Labs research center by
- Ken Thompson, Dennis Ritchie, and other engineers.

- Unix OS is designed to assist programming, text processing and many other tasks.

What is Linux ?

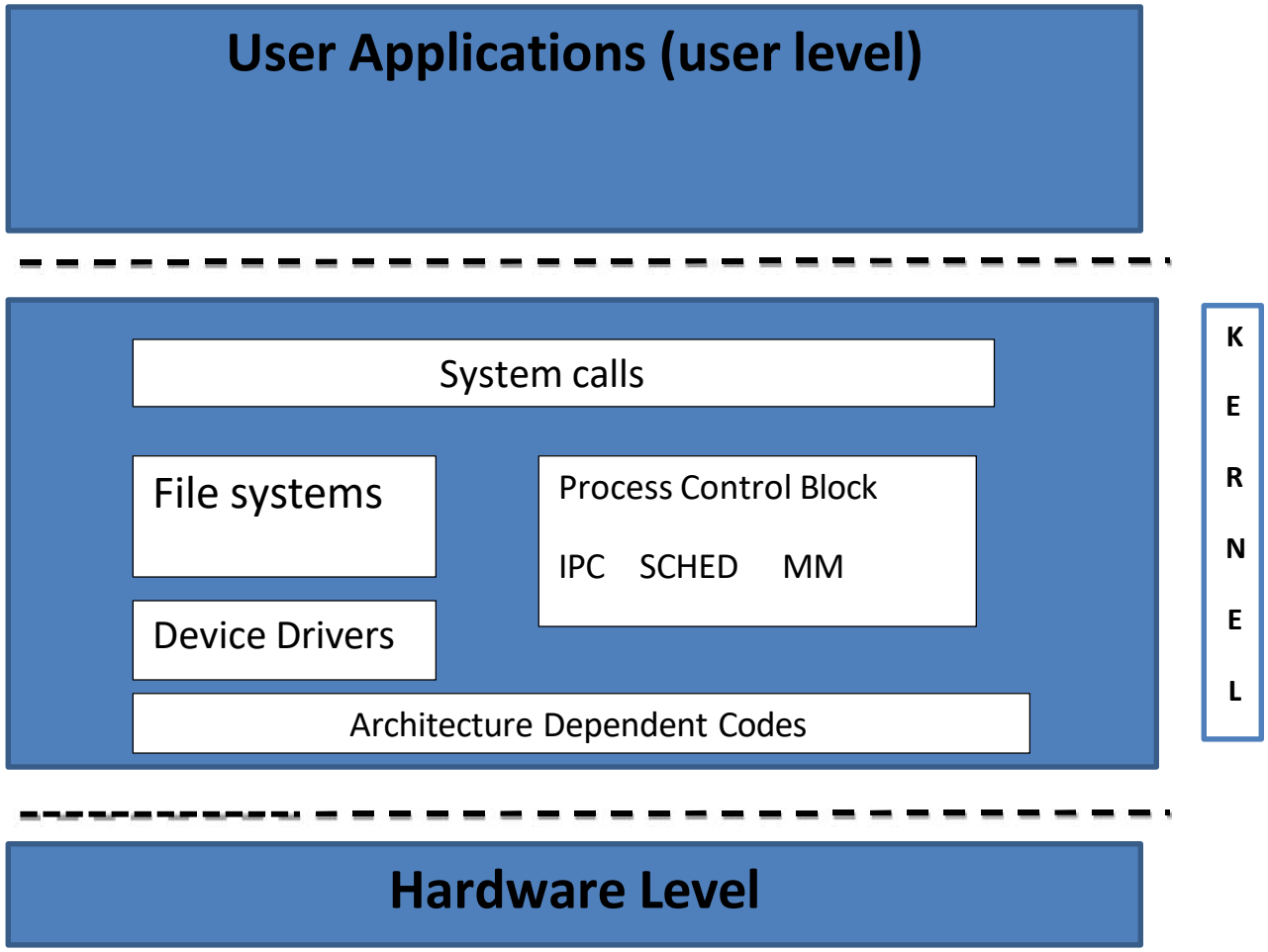
- The Linux is a Unix-like operating system.
- The Linux kernel was conceived and created in 1991 by Linus Torvalds.
- The Linux is a prominent example of free and open source software.
- Day-to-day development discussions take place on the Linux kernel mailing list (LKML).
- The Linux kernel is released under the GNU General Public License version2 (GPLv2),
- The Linux kernel is written in the C programming language supported by GCC (GNU Compiler Collection).

- The Linux is monolithic kernel design architecture.
- The Linux kernel provides several interfaces to user-space applications that are used for different purposes and that have different properties by design
- It supporting true preemptive multitasking (both in user mode and, since the 2.6 series, in kernel mode),
- virtual memory, shared libraries, demand loading, shared copy-on-write executable, memory management, the Internet protocol suite and threading.

The Linux kernel abstraction

- The Linux Kernel will treat everything as a file and process.
- which means that whatever instructions are comes from shell to kernel and hardware to kernel,
- It will treat as a file and process.
- File and process both are organized as a tree structure.

Architecture of Linux



About a shell ?

- A shell is a command-line interpreter (or) command line interface.
- Linux default login shell is bash (Bourne-Again SHell).
- user can interact with system through commands.
- All the commands are executed on the shell command line.
- We can shell is a parent (parent) of all user created commands(process).

- Shell is layered between user level and kernel level.

What is Shell Script ?

- Shell is a interface to kernel,it's interactive user commandline interface.
- Script is a file - it contains collection system commands in sequence manner.
- Shell will read the commands from file and executes it.
- Shell script file is an ordinary text file(or) ASCII file.
- Shell won't create any object file.

Types of shell

- There are several different shells available for Unix
- Bourne shell (sh)
- C shell (csh)
- TC shell (tcsh)
- Korn shell (ksh)
- Bourne Again SHell (bash)

To find all available shells in your system type following command:

[root@localhost ~]#

cat /etc/shells

- /bin/sh
- /bin/bash
- /sbin/nologin
- /usr/bin/sh
- /usr/bin/bash
- /usr/sbin/nologin

Shell Script

- Shell Script is a sequence of linux commands written in a text file(Script File) this is known as **shell script**.
- Shell Scripts allows you to automate these tasks for ease of use, reliability and reproducibility.
- Shell Scripts are interpreted not compiled.

Sha-Bang #!

- **#!/bin/bash** is Sha-Bang
- The above line says working shell is bash
- **#** Single line comment
- **<<ABC**

  Multiline comment

  ABC

Properties of good scripts

- A script should run without errors.
- It should perform the task for which it is intended.
- Program logic is clearly defined and apparent.
- A script does not do unnecessary work.
- Scripts should be reusable.

Variable

- A variable is a character string to which we assign a value.
- **Shell Support two types of variables**
  - User Defined Variable UDV
  - Shell (or) System Variables
- **UDV Syntax:-**
- variablename=value
- **Example:-**
- name="Mr.Karthik" # name is a variable , Mr.Karthik is value
- id=E101 # id is a variable, E101 is a value
- dept="sales" # dept is a variable ,sales is a value

Contd

- The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_).
- **Variable=value**
- Note :
- Assignment operator = LHS, RHS there is no space
- Var=10 # Valid
- **Var =10 # Error**
- **Var= 20 # Error**

- System variables - Created and maintained by Unix/Linux itself.
- This type of variable defined in CAPITAL LETTERS.

Rules for Naming variable name (Both UDV and System Variable)

- (1) Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character. For e.g. Valid shell variable are as follows

  HOME
  SYSTEM_VERSION
  vech
  no

- (2) Don't put spaces on either side of the equal sign when assigning value to variable. For e.g. In following variable declaration there will be no error

  $ no=10

  But there will be problem for any of the following variable declaration:

  $ no =10

```
$ no= 10
$ no = 10
```

- (3) Variables are case-sensitive, just like filename in Linux. For e.g.
  no=10
  No=11
  NO=20
  nO=2
  Above all are different variable name, so to print value 20 we have to use $ echo $NO and not any of the following
  $ echo $no # will print 10 but not 20
  $ echo $No# will print 11 but not 20
  $ echo $nO# will print 2 but not 20
- (4) You can define NULL variable as follows (NULL variable is variable which has no value at the time of definition) For e.g.
  $ vech=
  $ vech=""
  Try to print it's value by issuing following command
  $ echo $vech
  Nothing will be shown because variable has no value i.e. NULL variable.
- (5) Do not use **?,*** etc, to name your variable names.

Example

- System Variables or Shell Variables

PATH - Display lists directories the shell searches, for the commands.
HOME - User's home directory to store files.
PS1 - Display shell prompt in the Bourne shell and variants.
PS2 - Secondary prompt
MAIL - Path to user's mailbox.
PWD - Path to the current directory.
HOSTNAME -The system's host name
USER -Current logged in user's name.
SHELL -The current shell.
OSTYPE - Type of operating system.
MACHTYPE - The CPU architecture that the system is running on.
LOGNAME- Display login name

.
.

Exporting variables

- A variable created like the ones in the example above is only available to the current shell.
- It is a local variable: child processes of the current shell will not be aware of this variable.
- In order to pass variables to a subshell, we need to **export** them using the export built-in command.
- Variables that are exported are referred to as environment variables.

- **export VARNAME="value"**
- A subshell can change variables it inherited from the parent, but the changes made by the child don't affect the parent.

```
echo $$ → Current running Process ID
PID:1000
var=10
export var=10
bash   → create new shell
echo $$ → Current running process ID
PID:2000
echo $var → print 10
```

Example

```
 1     # Variable usages
 2     var1=10
 3     name="Mr.Ram"
 4     dept=sales
 5     place=chennai
 6     cost=1000

 7     echo "name"  # print name
 8     echo "$name" # print Mr.Ram

 9 # $name --->Scalar type variable or value (Single type)
10      echo "Name:$name"
11      echo "Department:$dept"
12      echo "Working place:$place"
13      echo "Cost is $cost"
```

Example

```
D1=`date`  # D1 is a variable
echo $D1
list=`ls`  # list is a variable
echo $list
echo "End of the line"
```

Example

```
# Employee Details
id=A101
name="Mr.Kumar"
dept="Sales"
place="pune"
B_pay=1234.565
echo "Employee Details..."
echo "------------------------------------------------------------------------"
echo "
     ID:$id
     Name:$name
     Department:$dept
     Place:$place
     BasicPay:$B_pay
"
echo "------------------------------------------------------------------------"
```

Example

```
echo "System Information:-"
echo "--------------------------------------------------------------------"
K=`uname`
KV=`uname -r`
S=$SHELL
SV=$BASH_VERSION
P=`pwd`
DATE=`date +%D`
name=`whoami`

echo " Working kernel name is:$K
      $K version is :$KV
      Working Shell name :$S
      version is $SV
      My name is $name
      Current date is $DATE
      my working path is $P
"
echo "End of the script.."
```

Example

- `echo "Enter your filename:"`
- `read fname`
- `F=$(ls -l $fname)`
- `echo "$F"`

read command

- The **read** command is useful in scripts when reading or asking an input from user.
- This **read** command is used when the script want to interact with user for his inputs to continue the script.
- **read** VARIABLE

- **read** VAR1 # Read a value from user input.
- **echo** $VAR1 # To display this value we have to use echo command.

Example

```
# Employee Details

echo "Enter Emp.Name"
read name  # name is a variable
echo "Hi..$name..Enter your ID"
read id    # id is a variable
echo "Enter $name working department"
read dept # dept is a variable
echo "Enter working place"
read place # place is a variable
echo "Enter B.Pay"
read B_pay # B_Pay is a variable
echo "Employee Details..."
echo "------------------------------------------------------------------------"
echo -e "ID:$id\t Name:$name\nDept:$dept\t Place:$place\n";
echo "------------------------------------------------------------------------"
```

Example

```
# student info
name="Mr.Kumar"
dept="computer science & engg"
s1=98
s2=80
s3=67
place="Bangalore"
echo " Student Information
************************************
        Name:$name
        Dept:$dept
        Sub1:$s1
        Sub2:$s2
        Sub3:$s3
        Place:$place
************************************
"
echo  # create one empty line
```

Example

- # System Information
- echo "working shell name is **$SHELL**"
- echo "working path name is **$PWD**"
- echo "My login name is **$LOGNAME**"
- echo "My login directory is **$HOME**"

Example

- # System Information using UDV
- **s1=$SHELL**
- **v1=$PWD**
- **v2=$HOME**
- **name=$LOGNAME**
- echo “

```
working shell name is $s1
working path is $v1
my log in name is $v2
my login name is $name “

# s1 ,v2 ,v2 and name are user defined variables(UDV)
```

Example

- To enable system command use backquotes `**command**` or **$(**command**)**
- echo "Today:$(date)"
- echo "Today:`date`"
- echo "Total No.of Users:$(who|wc -l)"
- echo "Total No.of Users:`who|wc -l`"

Example

- v1=$(date)
- v2=`date`
- v3=$(who|wc -l)
- v4=`who|wc -l`
- echo "Today:$v1"
- echo "Today:$v2"
- echo "Total No.of users:$v3"
- echo "Total No.of users:$v4"

- # v1,v2,v3 and v4 are user defined variables

Predict the out put of below scripts

Ex 1:

- no =10
- $no= 10
- echo $no
- echo $no

Ex 2:

- No=11
- NO=20
- echo $No
- echo $NO

Ex 3:

- v=
- v=""
- echo $v

Ex 4:

- How to Define variable x with value 100 and print it on screen?

Ex 5:

- How o Define variable **name** and **OS** name with value print it on screen?

Shell Operators

Shell Operators

- There are various types of operators supported by each shell

- 1. Arithmetic operators
- 2. Relational operators
- 3. Boolean operators
- 4. String operators
- 5. File test operators

Arithmetic Operators

- + , -, * , / are basic arithmetic operators.

Example :-

```
1.  echo " enter two value A and B "
2.  read A          # 10  as value A
3.  read B          #  20 as Value B
4.  echo " sum =  $A + $B  "
5.  echo " sub = $A - $B "
```

Now what do you think the output will be
Output is ?

Output ?

Here is your answer for that

sum = 10 + 20

sub = 10 - 20

Why ?

- Because echo just prints what is written with in the quote
- it can never identify any operators,it treat + - are like string not operators

Right way :

Method 1: -

```
sum = `expr  $A + $B`
echo $sum
```

NOTE : space between operators and operands

Method 2:-

- `sum = $((A+B))  # compound style`
- `echo $sum`

Example :1

```
1 echo  " enter  the two values for  A and B "
2 read A
3 read B
4 echo    Addition of A and B = `expr  $A  +  $B`
5 echo    Subtraction of A and B = `expr  $A  -  $B`
6 echo    Multiplication of A and B = `expr  $A  \*  $B`
7 echo    division of A and B = `expr  $A  /  $B`

# Line number 6 we used \* to avoid wild card(*)
# behaviour, \* is multiplication operator
```

Example : 2

```
1 echo   " enter the two vales A and B "
2 read A
3 read B
4 sum=`expr  $A  +  $B`
5 sub=`expr  $A  -  $B`
6 mul=`expr  $A  \*  $B`
7 echo " Addition :  $sum "
8 echo " Subtraction :  $sub "
9 echo " Multiplication :  $mul "
```

Example :3

```
1 echo  " enter the two vales A and B "
2 read A
3 read B
4 sum=$((A+B))
5 sub=$((A-B))
6 echo " Addition :  $sum "
7 echo " Subtraction :  $sub "
```

- using expr and compound (()) mode we can't compute floating point operation
- so we need to connect **bc** tool
- **what is bc ?**
- **bc – command line calculator**
- it is useful for performing mathmatical calculations
- bc support floating point and interger type arithmetic operations

bc screen

```
[root@localhost ~]# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free
Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
10+20
30
10.5+30.45
40.95
10.5*3+3.45
34.95
405/5
81
```

Example : 4

```
1 echo " enter the two vales A and B "
2 read A
3 read B
4 echo `echo $A  +  $B | bc`
5 echo `echo $A  \*  $B | bc`
```

Example : 5

```
 1 echo " Enter student name"
 2 read student
 3 echo " Enter $student marks"
 4 echo " Enter English marks out of 100 :"
 5 read eng
 6 echo " Enter Hindi marks out of 100:"
 7 read hin
 8 echo " Enter Physics marks out of 100 :"
 9 read phy
10 echo " Enter Math's marks out of 100 :"
11 read mat
12 echo " Enter Chemistry  marks out of 100 :"
13 read chem
14 sum=$((eng+chem+phy+mat+hin))
15 echo " total marks obtained by $student out of 500 : $sum"
16 avg=$((sum/5))
17 echo " Average  marks of $student is : $avg "
```

Relational Operators

- Relational operators are used to perform validation and testing purpose

- Two types of operation

1. Numerical based relational operators
2. String based relational operators

Numerical (Numbers) based relational operations.

-lt less than **-le** less than equal **-eq** equal

-gt greater than **-ge** greater than equal **-ne** not equal

Example

- a<b is equivalent to **$a -lt $b**
- a==10 is equivalent to **$a -eq 10**
- a>=b is equivalent to **$a -ge $b**
- a!=b is equivalent to **$a -ne $b**
- a<=b is equivalent to **$a -le $b**

Relational Operators

String based relational operators are :

< less than **<=** less than equal **=** equal

> greater than **>=** greater than equal **!=** not equal

- These all the relational operators are used in conditional statements and looping statements.

Boolean Tables

- logical AND opera ors -a
- logical OR operat rs -o
- logical NOT opera ors !

Logical Operators

- **-a (&&)**
- True -a True ==> True
- True -a False ==> False
- False -a True ==> False
- False -a False ==>False
- **-o (||)**
- True -o True ==> True
- True -o False ==> True
- False -o True ==> True
- False -o False ==>False
- **!**
- ! True =>False
- ! False => True

Rule of thumb:

- Use -a and -o inside square brackets,
- Use && and || outside.
- It's important to understand the difference between shell syntax and the syntax of the [command.
- && and || are shell operators.
- They are used to combine the results of two commands.
- Because they are shell syntax, they have special syntactical significance and cannot be used as arguments to commands.

Rule of thumb:

- [is not special syntax.
- It's actually a command with the name [, also known as test.
- Since [is just a regular command, it uses -a and -o for its and and or operators.
- It can't use && and || because those are shell syntax that commands don't get to see.

String Operators

- `[ "$v1" = "yes" ] && [ "$v2" != "Yes"]`
- The shell is evaluating the and condition
- `[ "$v1" = "yes" -a $v2 -lt 3 ]`
- `[[ $1 == "yes" &&  $v2 != "No" ]]`

Example

String based relational operators are :

< less than	**<=** less than equal	**=** equal
> greater than	**>=** greater than equal	**!=** not equal

- These all the relational operators are used in conditional statements and looping statements.

String Operators

- a="abc"
- b="efg"

- $a = $b
- $a = $b : a is equal to b

- $a != $b

- $a != $b : a is not equal to b

- -z $a

- "-z $a : string length is zero"

- -n $a

- "-n $a : string length is not zero"

File Test Operators

- Returns true if...
- -e
- file exists
- -f
- file is a regular file (not a directory or device file)
- -s
- file is not zero size
- -d
- file is a directory
- -b
- file is a block device
- -c
- file is a character device
- -p
- file is a pipe
- f1 -nt f2
- file f1 is newer than f2
- f1 -ot f2
- file f1 is older than f2
- f1 -ef f2
- files f1 and f2 are hard links to the same file
- !
- "not" -- reverses the sense of the tests above (returns true if condition absent).

Conditional Statements

Conditional statements

- Shell support two types of conditional statement
- 1.single conditional statement using **if** statement
- 2. Multi conditional statement using **case** statement

General **if** statement behavior

- At times you need to specify different courses of action to be taken in a shell script, depending on the success or failure of a command.
- The **if** construction allows you to specify such conditions.

The most compact syntax of the if command is:

```
if TEST-COMMANDS
then
        CONSEQUENT-COMMANDS
 fi
```

The TEST-COMMAND list is executed, and if its return status is zero, the CONSEQUENT-COMMANDS list is executed.

The return status is the exit status of the last command executed,or zero if no condition tested true.

test command and [ ]

- The TEST-COMMAND often involves numerical or string comparison tests, but it can also be any command that returns a status of zero when it succeeds and some other status when it fails.
- if [ ] this is built in **test** operator
- we can use **test** command,instead of using **[]**

Conditional Statement

- we can write if statement 3 different style as follows

- 1. if only
- 2. if ..else
- 3. If ..elif..else

1. if only syntax: -

```
if [ conditional statement ]   # test the condition
then
        TRUE BLOCK
fi                             # fi end of the if statement
```

[ ] operator will evaluate the condition statement , if will determined condition status (true or false)

using **test** command

- We can write using test command
- if **test** condition

  then

  TRUE BLOCK

  fi

Conditional Statement

if ..else syntax: -

```
if  [ conditional statement ]
then
      TRUE BLOCK
else
      FALSE BLOCK
fi
```

Conditional Statement

if...elif ..else ..fi syntax:-

```
if [ conditional statement ]
then
   TRUE BLOCK 1
elif [ conditional statement ]
then
   TRUE BLOCK 2
elif [ conditional statement ]
then
   TRUE BLOCK 3
else
   FALSE BLOCK
fi
```

Example :1
if only style

```
echo "Enter A and B value:"
read a;read b
if [ $a -lt $b ]
then
    echo "True..$a < $b"
fi
echo "End of the script.."
```

```
1 echo " enter the value of less than 10 "
2 read N
3 if  [  $N  -lt  10 ]
4 then
5 echo "True : $N is less than 10 "
6 fi
```

Example : if ..else ..style

```
echo "Enter A and B value:"
read a;read b
if [ $a -lt $b ]
then
   echo "True..$a < $b"
else
   echo "False..$a > $b"
fi
echo "End of the script.."
```

```
echo "Enter Enquiry No"
read eno
if [ $eno -ge 100 ]
then
    echo "$eno is valid entry.."
    echo "Enter your vendor code"
    read vno
    if [ $vno -ge 500 ]
    then
        echo "$vno is valid vendor code"
    else
        echo "Sorry ..$vno is invalid vendor code.."
    fi
else
    echo "The $eno is not valid Enquiry no.."
fi
```

Example :

```
echo -n "Enter Quotation Number:"
read qno
if [ $qno -ge 100 -a $qno -le 500 ]
then
        echo "The $qno is valid quotation number"
        echo "Enter your PO number:"
        read po
        if [ $po -ge 500 -a $po -lt 600 ]
        then
                echo "The $po is valid entry.."
                echo "Enter vendor code and name"
                read vno;read name
                echo "Enter Item details.."
                read item
                echo "We received $item on `date +%D`"
                echo "The $item details:-
                      Quotation No:$qno
                      PO No:$po
                      Vendor Code:$vno and Name:$name"
        else
echo "Sorry $po is not valid PO Number.."
        fi
else
        echo "Sorry $qno is not valid quotation number.."
fi
```

```
 1 echo " Enter the student name "
 2 read name
 3 echo "Enter $name place"
 4 read place
 5 echo "Enter 3 subject marks: out of 100"
 6 read s1
 7 read s2
 8 read s3
 9 echo "------------------------------------------------------------------------"
10 echo "$name information:
```

```
11        Name : $name
12        Place : $place
13        S1 : $s1
14        S2 : $s2
15        S3 : $s3 "
16 sum=`expr $s1 + $s2 + $s3`
17 avg=`echo $sum/3 | bc`
18 echo "Total: $sum"
19 echo "Average: $avg"
20 if [ $s1 -ge 50 -a $s2 -ge 50 -a $s3 -ge 50 ]
21 then
22          echo "Result: PASS"
23 else
24          echo "Result: FAIL"
25 fi
```

Example :
if ..else ..elif style

```
 1 echo "Enter A value and B value"
 2 read a
 3 read b
 4 if [ $a -eq $b ]
 5 then
 6         echo "True : $a and $b are equal"
 7 elif [ $a -gt $b ]
 8 then
 9         echo "True : $a greater than  $b "
10 elif [ $a -lt 10 ]
11 then
12         echo "True : $a less than $b "
13 else
14         echo "Else B oc  : $a < $b‟
15 fi
```

```
1 if [ $UID -eq 0 ];then
2             echo "Your root user"
3 else
4         echo "Your not root user"
5         echo "Hi..$LOGNAME your id is $UID"
6 fi
```

Example

```
# This script will test if we're in a leap year or not.


year=`date +%Y`
if(( ("$year" %400)=="0"))||(( ("$year" %4 =="0") && ("$year" % 100 != "0")
));then
        echo "This is a leap year."
else
        echo "This is not a leap year"
fi
```

String comparation

```
echo -n "Enter your login name:"
read name
echo "Hi..$name Enter your Password"
read -s p1
echo "Re-type your Password.."
read -s p2

if [ $p1 = $p2 ]
then
    echo "Hi..$name your valid login user.."
else
    echo "Sorry $name your login is failed.."
fi
```

Example
logical operator (-a)

```
echo "Enter 3 subject Marks.."
read s1;read s2;read s3
if [ $s1 -gt 100 ];then
      echo "the max marks obtained is 100"
      exit
elif [ $s2 -gt 100 ];then
      echo "the max marks obtained is 100"
      exit
elif [ $s3 -gt 100 ];then
      echo "the max marks obtained is 100"
      exit
fi
if [ $s1 -ge 50 -a  $s2 -ge 50 -a $s3 -ge 50 ];then
      echo "Result:PASS"
else
      echo "Result:FAIL"
fi
```

logical operator (-o)

```
read s1;read s2;read s3
if [ $s1 -gt 100 -o $s2 -gt 100 -o $s3 -gt 100 ];then
     echo "the max marks obtained is 100"
     exit
fi
if [ $s1 -ge 50 -a  $s2 -ge 50 -a $s3 -ge 50 ];then
     echo "Result:PASS"
else
     echo "Result:FAIL"
fi
```

Example

logical operator (!)

```
if [ 100 -eq 100 ]
then
      echo "True.."              # print True
else
      echo "Fail"
fi
sleep 3
echo "using not operation.."
if ! [ 100 -ne 100 ]
then
      echo "True.."              # print True
else
      echo "Fail.."
fi
```

Example

```
a=0;b=0
# do something else with a or b
if [[ $a -eq 2 ]] || [[ $b -eq 4 ]]
then
    echo "a or b is correct"
else
    echo "a and b are not correct"
fi
```

Example

```
if((10==10))&&((10!=20))
then
    echo "true.."
else
    echo "Fail.."
fi
```

Example

```
if [ 10 -eq 10 ] && [ 10 -ne 20 ]
then
    echo "true.."
else
    echo "Fail.."
fi
```

Example

```
if [ 10 -eq 10 -a 10 -ge 5 ] && [ 10 -ne 20 ]
then
    echo "true.."
else
    echo "Fail.."
fi
```

```
echo "Enter File name:"
read fname
test -e $fname

if [ $? -eq 0 ]
then
    echo "The $fname is available"
else
    echo "The $fname is Not Available"
fi
```

Example

```
echo "Enter File name:"
read fname
test -e $fname

if [ $? -eq 0 ]
then
      echo "The $fname is available"
      test -f $fname
      if [ $? -eq 0 ];then
               echo "The $fname is Reg.file"
      fi
      test -d $fname
      if [ $? -eq 0 ];then
               echo "The $fname is Directory.."
      fi

else
      echo "The $fname is Not Available"
fi
```

Example

```
echo "Enter File name:"
read fname
test -e $fname

if [ $? -eq 0 ]
then
        echo "The $fname is available"
        test -f $fname
        if [ $? -eq 0 ];then
                echo "The $fname is Reg.file"
        else
                echo "The $fname is Not Reg.file"
        fi
        test -d $fname
        if [ $? -eq 0 ];then
                echo "The $fname is Directory.."
        else
                echo "The $fname is Not Directory file"
        fi

else
        echo "The $fname is Not Available"
fi
```

Example
using **[ ]** operator

```
echo "Enter your input file name:"
read fname
if [ -e $fname ];then
    echo "The $fname is available"
else
    echo "The $fname is not available"
fi
```

Example using [ ] operator

```
echo "Enter your input file:"
read fname
if [ -f $fname ];then
     echo "The $fname is Reg.file"
elif [ -d $fname ];then
     echo "The $fname is Directory File"
elif [ -c $fname ];then
     echo "The $fname is char type device file"
elif [ -b $fname ];then
     echo "The $fname is Block type device file"
elif [ -p $fname ];then
     echo "The $fname is pipe type file"
else
     echo "The $fname is not available"
fi
```

case statement

- **case** statement is generally used as a shortcut for writing **if/else** statements.
- The **case** statement is always preferred when there are many items to select from instead of using a large **if/elif/else** statement.
- The **case** statement is terminated with **esac** (case backwards).

Example

```
echo "Enter your book name:"
read book

case $book in
"unix")  echo "
                Book Name:$book
                Author Name:Mr.X
                Price :456INR"
          ;;

Linux)  echo "
                Book Name:$book
                Author Name:Mr.Y "
        ;;
aix)    echo "Your input book is $book" ;;
minix)  echo "Book name:$book
                Vol:3.45"
        ;;
*)      echo "the input book :$book is not available"
esac
```

Example

```
echo "Enter your dept code:"
read ch

case $ch in
s)  echo "Sales.." ;;
p)  echo "Production" ;;
F)  echo "FI" ;;
a)  echo "Accounts.." ;;
*)  echo "$ch is invalid Dept code.."
esac
```

Example

```
echo "Enter your dept code:"
read ch

case $ch in
S|s)     echo "Sales.." ;;
p| )     echo "Production" ;;
F| )     echo "FI" ;;
a| )     echo "Accounts.." ;;
*) echo "$ch is invalid Dept code.."
esac
```

```
echo "Enter your dept code:"
read ch

case $ch in
S|s)        echo "Sales.." ;;
p|P)        echo "Production" ;;
F|f)        echo "FI" ;;
a|A|b|D) echo "Accounts.." ;;
*)  echo "$ch is invalid Dept code.."
esac
```

Example

```
echo "Enter your OS"
read os
case $os in
"unix"|"linux"|aix)   echo "Unix type os" ;;
"win") echo "Windows OS" ;;
"bash"|"sh"|ksh)      echo "Support interface scripting.." ;;
tcsh|csh|expert)      echo "support FTP automation.." ;;
*)  echo "$os is invalid os";
esac
```

```
echo -e "\tSystem Information:-"
echo -e "\t************************"
echo "
        1.Display your working kernel name
        2.Display your Shell name
        3.Login name
        4.Today Date
        5.Current working Directory path
"
echo -e "\t************************"
echo -n "Enter your Option:" ;read n

case $n in
1)      echo "Working kernel name is $(uname)
              Version is $(uname -r)"
        ;;
2)      echo "Working Shell is $SHELL
              Version is $BASH_VERSION"
        ;;
3)      echo "My login name:$LOGNAME and Login id is $UID" ;;
4)      echo "Today:`date +%D`" ;;
5)      echo `pwd` ;;
*)      echo "Sorry $n is invalid option..select from [1 to 5]"
esac
```

```
echo "Enter var1 and var2:"
• read v1;read v2
• echo "Enter Arth.operator:"
• read opt

• case $opt in
•     +)    echo `echo $v1 + $v2|bc` ;;
•     -)    echo `echo $v2 - $v1|bc` ;;
•     \*)   echo `echo $v1 \* $v2|bc` ;;
•     *)    echo "$opt is invalid operator.."
• esac
```

Thank you